Processing Geo-Dispersed Big Data in an Advanced MapReduce Framework

Hongli Zhang, Qiang Zhang, Zhigang Zhou, Xiaojiang Du, Wei Yu, and Mohsen Guizani

Abstract

Big data has emerged as a new era of information generation and processing. Big data applications are expected to provide a lot of benefits and convenience to our lives. Cloud computing is a popular infrastructure that has the resources for big data processing. As the number of mobile devices is fast increasing, mobile cloud computing is becoming an important part of many big data applications. In this article, we propose a novel MapReduce-based framework to process geo-dispersed big data in mobile cloud architecture. The proposed framework supports simple as well as complex operations on geo-dispersed big data, and uses various data aggregation schemes to satisfy different application requirements.

Big data takes many forms, including messages in social networks, data collected from various sensors, captured videos, and so on. Big data applications aim to collect and analyze large amounts of data, and efficiently extract valuable information from the data. A recent report shows that the amount of data on the Internet is about 500 billion GB. With the fast increase of mobile devices that can perform sensing and access the Internet, large amounts of data are generated daily. In general, big data has three features: large volume, high velocity and large variety [1]. The International Data Corporation (IDC) predicted that the total amount of data generated in 2020 globally will be about 35 ZB. Facebook needs to process about 1.3 million TB of data each month. Many new data are generated at high velocity. For example, more than 2 million emails are sent over the Internet every second.

Mobility services such as Google Maps and Navigation Service provide benefits and convenience to people. These applications are big data applications because the data set size is big and the data update rate is fast [2]. Large amounts of fresh mobility-related data are generated every day, for instance, video surveillance data collected by high-definition cameras at roadsides and junctions. Typically, the rapidly generated big data are not uploaded to a data center at once. Instead, the fresh big data is quickly stored in local servers temporarily. Previous research works on big data mainly study efficient processing techniques and analytical methods for big data in a clustered environment, and do not consider a geo-dispersed big data scenario. The above transportation service based on fresh and historical big data belongs to a geo-dispersed big data scenario. In this situation, it is a challenge to efficiently handle a request for geo-dispersed big data application. In addition, different service targets require different complexities of operations on big data. In general, operations on big data can be divided into two categories: simple operations and complex operations. For example, retrieval belongs to simple operations, while analysis of video content (based on data mining) is a complex operation. A framework for efficiently processing geo-dispersed big data should support both simple and complex operations.

Mobile cloud computing [3–5] is an emerging cloud service model based on mobile computing and cloud computing. As the computing capability of mobile devices increases, mobile cloud computing can organize and utilize computation resources of distributed mobile devices. A new model for mobile cloud computing is called the cloudlet-based mobile cloud model. The cloudlets [6, 7] are deployed near Wi-Fi access points (APs) and cellular base stations to provide cloud services efficiently, and decrease the network cost between mobile users and a central cloud. In the mobile cloud architecture, there are some scenarios where large fresh data sets (as part of big data) are generated rapidly and quickly stored in cloudlets, and the fresh data are migrated to the central cloud periodically. In this situation, if some requests for big data applications come in, the conventional method of uploading a large amount of fresh data to the data center is not efficient in terms of communication overhead and response time. Most previous research works on mobile cloud computing discuss how to efficiently offload tasks from mobile devices to a cloudlet or a central cloud in order to save mobile device energy and reduce task completion time. However, few works study how to utilize mobile cloud computing to process geo-dispersed big data and optimize response time to mobile users. To efficiently process geo-dispersed big data using mobile cloud computing, collaboration among nodes is important.

In this article, we propose a novel and flexible framework based on MapReduce to support simple as well as complex operations on geo-dispersed big data. The proposed frame-

Hongli Zhang, Qiang Zhang, and Zhigang Zhou are with Harbin Institute of Technology.

Xiaojiang Du is with Temple University.

Wei Yu is with Towson University.

Mohsen Guizani is with Qatar University.

work is referred to as the advanced MapReduce framework (AMF). For a request with simple operations, AMF employs cooperative processing in the mobile cloud and MapReduce to process geo-dispersed big data. First, the proposed framework automatically divides a big job into several branch jobs according to the distribution of input data, and then each branch job is performed using cooperative processing in the mobile cloud. For a request with complex operations, AMF extracts the required multiple inputs from geo-dispersed big data in parallel, and then aggregate extracted required multiple inputs from different cloud nodes. After the aggregated data are processed by performing complex operations, AMF creates the final results and sends them to the user. For complex operations, AMF uses different data aggregation schemes for different application requirements. For real-time applications, the goal is to minimize the response time to mobile users. For non-real-time applications, AMF makes a trade-off between response time and communication overhead. And the proposed aggregation schemes are based on collaboration among cloud nodes. AMF adaptively utilizes MapReduce to perform simple operations on geo-dispersed big data by distributed and parallel computing. Moreover, AMF improves MapReduce to support complex operations on geo-dispersed big data by aggregation schemes.

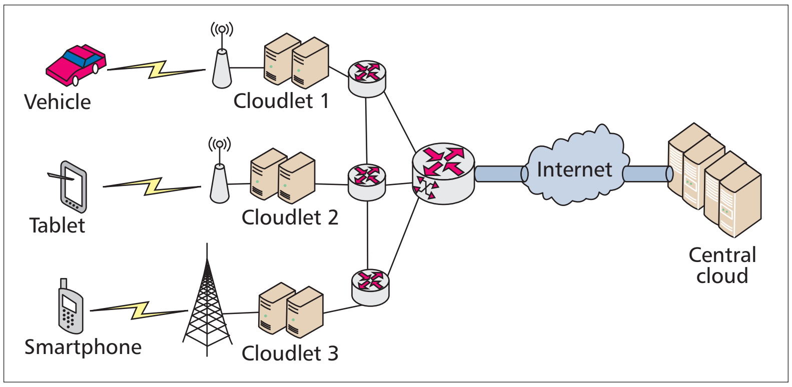

Figure 1. Mobile cloud architecture.

When MapReduce Meets Geo-Dispersed Big Data

MapReduce is a software framework introduced by Google to perform distributed computation on large data sets. MapReduce [8] is a promising computing model for big data processing. The MapReduce framework has been used widely by many corporations such as Google, Yahoo, and Amazon to process big data efficiently. The main idea of the MapReduce framework is to split a large job into a number of smaller tasks, including mapping and reducing tasks, and these tasks are performed independently on different worker nodes. Before starting map tasks, input data need to be partitioned into several small data blocks of the same size ranging from 16 to 64 MB. Each data block is then assigned by a master to a worker along with a map operation. The mapper (i.e., the worker assigned to a map task) applies a map operation to compute intermediate key-value pairs. A master is in charge of assigning map and reduce tasks to workers. A map operation consists of three functions: map function, sort function, and combine function. The map function can be obtained from the specific operation corresponding to a request. The sort function is responsible for sorting the intermediate values computed by mappers in order to group key-value pairs corresponding to the same key. The combine function is utilized to integrate all the intermediate values sharing the same key so that the size of the intermediate values is reduced. Then the intermediate values are partitioned into R blocks by a hash function and stored in local disks. In addition, a reduce operation includes three functions: shuffle, merge, and reduce. The shuffle function enables each reducer to pull its intermediate values from local disks. The merge function groups all intermediate values sharing the same key. The reduce function implements the requested simple operation on input data.

Typical work environments for MapReduce are clustered environments in which many machines have stable connectivity, high bandwidth, and a shared file system. When all of the big data is stored in a single data center, the MapReduce framework is simple, flexible, and efficient. However, the huge amount of distributed real-time information introduces a new scenario. In the new scenario, fresh data as a part of big data stored in cloudlets are geographically separated from data in the central cloud, and migrating a large amount of fresh data to the central cloud may cause a large delay to users. Hence, the conventional MapReduce framework for clustered environments is not suitable for the above scenario in terms of network delay. Moreover, operations on large data sets supported by MapReduce are usually simple mathematical operations such as count, sort, and selection. At present, MapReduce does not support complex operations (e.g., data mining and data analysis) on big data very well. Hence, the issue of efficiently performing complex operations on geo-dispersed big data in the mobile cloud model needs to be solved.

Mobile Cloud Architecture for Geo-Dispersed Big Data Applications

Figure 1 shows the mobile cloud architecture that is used to provide better support for geo-dispersed big data applications. The architecture consists of several cloudlets and a central cloud. The central cloud stores part of the big data, and the cloudlets have large amounts of fresh data (part of the big data), which are uploaded to the central cloud periodically to update the data set. The central cloud has sufficient computation resources to process all of the big data. However, migrating large amounts of fresh data from cloudlets to the central cloud may cause long delays. On the other hand, a cloudlet has less computation resources than the central cloud but very short communication delays to mobile users. For some geo-dispersed big data applications that require complex mathematical operations, the corresponding multiple inputs cannot be partitioned and processed by distributed and parallel computing. Currently, complex mathematical operations are not well supported by conventional MapReduce [9]. In this situation, when a request for a geo-dispersed big data application happens, migration of a large amount of data is not efficient in terms of response time. Hence, the cloudlets should be utilized to assist in performing complex operations on geo-dispersed big data and reduce response time.

In the mobile cloud environment, mobile devices play a key role in generating big data and requesting big data applications. Mobile devices discussed in this article include vehicles and small smart mobile terminals, including tablets, smartphones, and so on. For vehicles, there are some cases in which data collection rates can outperform the Internet, such as video surveillance in buses. At present, many buses have installed high-definition camera systems to monitor in-bus conditions. Traditionally, every bus needs a very large-volume hard drive to store video content for a few days. Then the

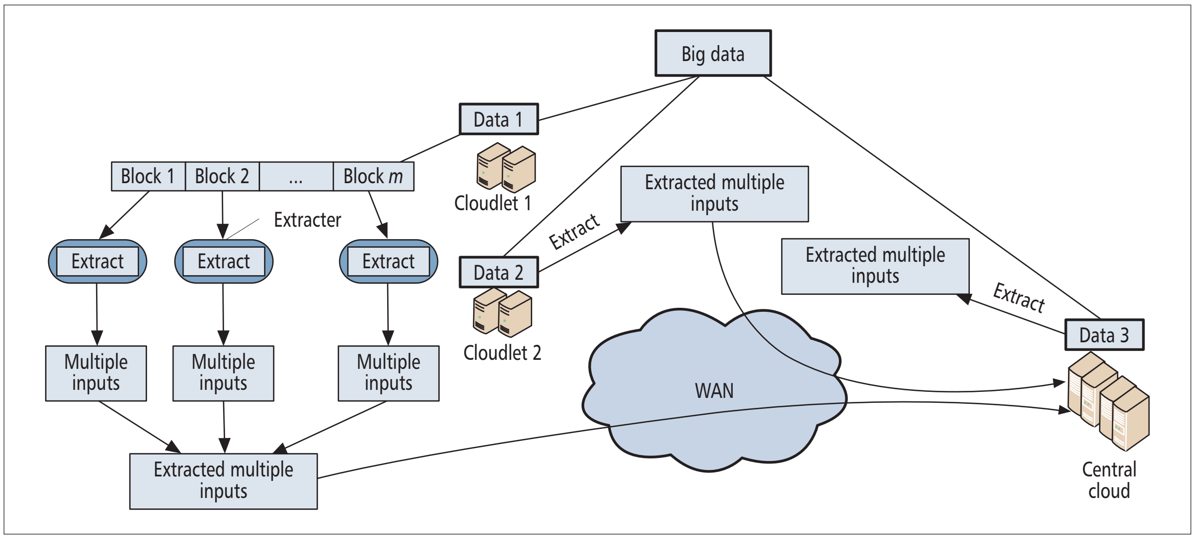

Figure 2. Extraction module and aggregation operations in AMF.

video content is checked in an offline manner [5]. In mobile cloud architecture, cloudlets can be used to store the rapidly generated video surveillance content to implement timely video content processing and save the high cost of large-volume hard drives. In this case, preprocessing using cloudlets can reduce the communication delay for delivering a large amount of data to a central cloud for complex operations. In addition, for simple operations, distributed and parallel computing using cloudlets and a central cloud can decrease the response time of a request.

This mobile cloud architecture is flexible and advantageous in supporting geo-dispersed big data applications for mobile users. First, the architecture contains two layers of clouds that are in different locations. The central cloud is usually distant from mobile users. Cloudlets are deployed near mobile users to provide services quickly. For instance, a mobile user can collect and upload surrounding real-time information to a nearby cloudlet, and then other mobile users arriving at the same or neighboring cloudlet are able to retrieve and download recent data fast. Second, cloudlets reduce the workload of the central cloud so that tasks in the central cloud can be performed faster. Third, a seamless connection between mobile devices and cloudlets can be accomplished by hybrid wireless communication technologies such as Wi-Fi and cellular 3G/4G. For example, a mobile device equipped with standard wireless interface can access to a nearby cloudlet through Wi-Fi AP or cellular base stations.

Features of Complex Mathematical Operations on Geo-Dispersed Big Data

Complex mathematical operations on normal data can be abstracted as a complex algorithm that can process multiple inputs. We discuss the features of complex mathematical operations on geo-dispersed big data in the following. First, finding multiple required inputs is critical for performing complex mathematical operations efficiently on big data because many irrelevant data are included in the original input data. In addition, the original input data that contain the multiple required inputs are very likely located in different clouds due to geo-dispersed big data. Therefore, aggregation of geo-dispersed multiple inputs needs to be accomplished in order to obtain the complete inputs. Aggregation in this article means transferring geo-dispersed multiple required inputs to a cloud that can perform the requested complex algorithm with complete inputs. Second, a cloud performing the complex algorithm on multiple inputs should guarantee that the entire operating procedure of the complex algorithm runs correctly. Due to some inherent attributes of complex mathematical operations, partitioning multiple inputs or the algorithm into several parts for parallel computing usually does not work. Hence, given all inputs, a complex algorithm that can process the inputs and create results correctly should be performed in one machine rather than a cluster of machines.

The Advanced MapReduce Framework

In this article, we propose a novel and flexible framework, AMF, based on MapReduce to process geo-dispersed big data. Different from the conventional MapReduce framework in a clustered environment, AMF focuses on supporting complex mathematical operations and efficiently performing simple operations on geo-dispersed big data. AMF combines cooperative processing in a mobile cloud with MapReduce to efficiently process geo-dispersed big data. In this article, we mainly consider the case in which partial or complete large volumes of input data are promptly stored in cloudlets. In addition, we assume that both cloudlets and the central cloud have sufficient computation resources to perform simple and complex operations. The specific method of AMF is laid out below.

Performing Complex Operations in AMF

AMF mainly utilizes distributed extracting to decrease the size of data that needs to be aggregated and processed. First, multiple required inputs need to be extracted from large amounts of original input data so that the size of input data is reduced. In other words, extracting multiple required inputs means refining original input data. Due to the reduced size, the time to transfer the refined input data is less than that of the original input data. AMF employs distributed and parallel computing to extract multiple required inputs from the original input data (geo-dispersed big data). Second, the multiple inputs are aggregated to guarantee the complete inputs. Third, the requested complex mathematical operations are performed in a cloud node.

As shown in Fig. 2, AMF automatically divides the entire extraction job into several branch jobs by distributed extracting in each cloud. A branch job consists of extract operations and the original input data. Before starting the extract task, the original input data needs to be partitioned into several small data blocks of the same size, and the size of a data block is bigger than the size of the minimal input unit. Each data block is then assigned to an extractor along with an extraction operation. The extractor applies the extraction function to obtain some of the required inputs. The extraction function can be obtained from the specific request for a big data application. After the entire extraction job is done, an aggregation of the multiple required inputs needs to be performed to create the complete inputs. Once the aggregation of the geo-dispersed multiple inputs is done in a cloud, the corresponding cloud starts to perform complex mathematical operations on the inputs. As shown in Fig. 3, the complete inputs cannot be partitioned and are sent to an analyzer directly. The analyzer applies an analysis function to obtain the final results. The analysis function is an algorithm with complex mathematical operations.

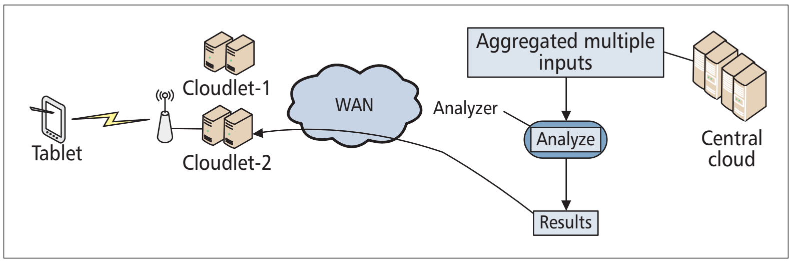

Figure 3. Analysis module in AMF.

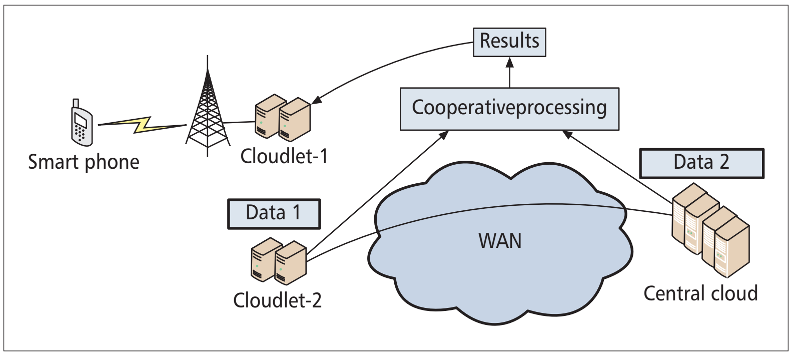

Figure 4. Simple operations in AMF.

Performing Simple Operations in AMF

As shown in Fig. 4, the proposed framework employs cooperative processing in mobile cloud and MapReduce to perform simple operations on geo-dispersed big data. First, the proposed framework automatically divides a big job into several branch jobs according to the distribution of input data, and then each branch job is performed using cooperative processing in the mobile cloud. The main idea of cooperative processing is to leverage a central cloud to accelerate processing and reduce response time. When a cloud node starts to perform simple operations on input data, MapReduce is utilized to perform computing in parallel. We divide simple operations into two categories. For simple operations in the first category, the distributed intermediate results do not need to be aggregated for further processing such as search operation. For the second category, the distributed intermediate results need to be aggregated and processed further for instance sort operation.

First, we demonstrate our method of performing simple operations in the first category. Considering the fact that the data sizes of the processed results are normally small, the communication delay for delivering the final results is negligible compared to the computing time of the large amount of input data. As the intermediate results in the first category are part of the final results, the corresponding communication delay is also negligible. The proposed cooperative processing makes cloudlets allocate a number of chunks of input data to the central cloud. In other words, the total tasks are divided and processed in parallel by cloudlets and the central cloud. While a cloudlet is transmitting input data to the central cloud, the cloudlet is also processing the input data except the data allocated to the central cloud. When both the cloudlets and the central cloud finish their own processing, the complete final results are created. Our method calculates the amount of data chunks allocated to the central cloud in order to reduce the total response time. We give notation definition in our scheme as follows.

Let a denote the time to execute the simple operations on a single chunk of data in a cloudlet, b denote the time to execute the simple operations on a single chunk of data in the central cloud, and c denote the time to transfer one single chunk from a cloudlet to the central cloud. If the total amount of chunks of input data in a cloudlet is m, and the number of chunks allocated to central cloud is n, the time to process $(m - n)$ chunks using the cloudlet is denoted as T_1^p, and the total time to transmit n chunks through a WAN and process them using the central cloud is denoted as T_2^p. Thus, for performing simple operations on input data in a cloudlet, the total response time of cooperative processing is:

$$T^c = \min(\max(T_1^p, T_2^p)). \quad (1)$$

In Eq. 1, $T_1^p = (m - n)a$ and $T_2^p = nc + nb$. When

$$n = \frac{ma}{a+b+c},$$

the minimal value of $\max(T_1^p, T_2^p)$ can be acquired. When big data is quickly stored in distributed cloudlets, each cloudlet independently calculates its own T^c and performs simple operations by cooperative processing. Once some intermediate results are generated in a node, they are directly transmitted to mobile users as a part of the final results.

Second, we demonstrate our method to perform simple operations in the second category. Like the method in the first category, the proposed cooperative processing makes cloudlets allocate a number of chunks of input data to the central cloud to implement parallel computing. When both the cloudlets and the central cloud finish their own processing, the aggregation of intermediate results is started. Our method calculates the amount of chunks of data allocated to the central cloud and the corresponding minimal aggregation time in order to reduce the total response time. We give notation definition in our scheme as follows.

Let G_1 denote the data size of intermediate results corresponding to $(m - n)$ chunks in the cloudlet, and G_2 denote the data size of intermediate results corresponding to n chunks in the central cloud. Let T_1^a denote the time to aggregate intermediate results and create the final results in the cloudlet, and T_2^a denote the time to aggregate intermediate results and create the final results in the central cloud. For other notations, we still use the above definition such as a, b, and c. Considering the fact that the data sizes of the final results are normally small, the communication delay for delivering the final results is negligible. Equation 2 demonstrates our

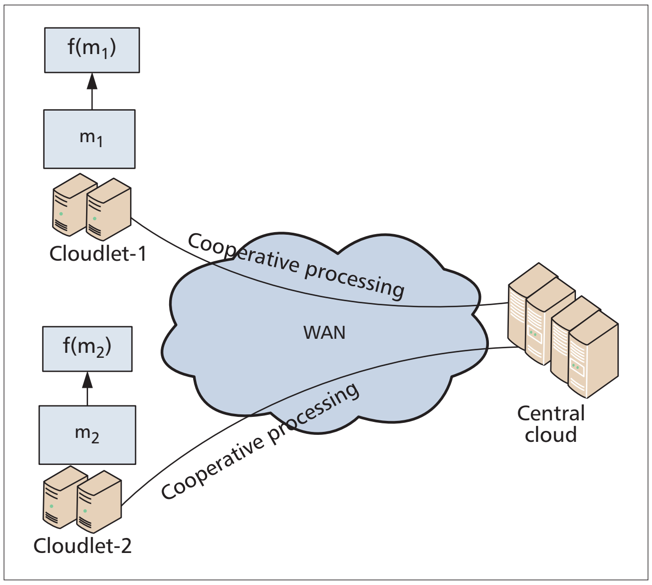

Figure 5. Response time using cooperative processing.

method of calculating the response time in the case where the complete input data is only stored in a cloudlet. So the corresponding total response time of cooperative processing is

$$T^c = \min(\max(T_1^p, T_2^p)) + \min(T_1^a, T_2^a). \qquad (2)$$

In Eq. 2, the minimal value of $\max(T_1^p, T_2^p)$ can be acquired as in Eq. 1. Let s denote the size of a chunk. Hence,

$$T_1^a = \frac{G_2 c}{s} + \frac{(G_1 + G_2)a}{s}$$

and T_2^a can also be obtained. We use a function $f(x)$ to describe the relationship between the size of the processed results and the size of the input data, which is denoted as x. For different applications, the corresponding $f(x)$ is different. Machine learning can be used to establish the function $f(x)$. In this article, we do not specifically discuss how to establish $f(x)$, but give a method to calculate the sizes of the processed results of input chunks $(m - n)$ in the cloudlet and input chunk n in the central cloud (i.e., G_1 and G_2). Therefore, through the initial values m, s, $f(x)$, and n, we can calculate the values of G_1 and G_2. Then, $\min(T_1^a, T_2^a)$ can be obtained. Thus, the total response time T^c of cooperative processing can be calculated when a request with simple operations happens. At last, we compare the response time corresponding to only using the cloudlet ma with the T^c, the processing method corresponding to the lower response time is selected and performed. For distributed input data stored in several cloudlets, we demonstrate our methods to perform simple operations in the second category as follows.

We consider the case in which the complete input data is stored in two cloudlets, and the cloudlets have different input data corresponding to m_1 chunks and m_2 chunks. Let a_i denote the time to execute the simple operations on a single chunk of data in cloudlet i, T_i^c the total response time calculated by Eq. 2 corresponding to cloudlet i, and d denote the time to transfer one single chunk from one cloudlet to another. For other notations, we still use the above definitions such as b and c. First, each cloudlet independently calculates its own T_i^c and $m_i a_i$ by our above method and determines the processing method according to the respective response time. After the first step, intermediate results corresponding to input data from different cloudlets are created, and they may be distributed at different nodes due to cooperative processing. Second, according to the distribution of the intermediate results, calculate the aggregation time in each candidate aggregation node i which has partial intermediate results or is the central cloud. The calculation of total response time can be done in the central cloud or cloudlets. Before the calculation, the initial values including the function $f(x)$ need to be shared among these computing nodes. Based on various aggregation times and processing times in different aggregation nodes, the respective total response time T_i^b can be obtained. Taking the case in Fig. 5 as an example, where the intermediate results $f(m_1)$ and $f(m_2)$ are created in cloudlets 1 and 2, respectively, using Eq. 2, the total response time corresponding to aggregation node cloudlet 1 is

$$T_1^b = \max(T_2^c + f(m_2)d, T_1^c) + (f(m_1) + f(m_2))a_1$$

For brevity, the total response times, T_2^b and T_3^b, corresponding to aggregation node cloudlet 2 and the central cloud, respectively, are not demonstrated in detail. At last, the processing method and the aggregation node corresponding to the response time $\min(T_1^b, T_2^b, T_3^b)$ are performed and selected. For other cases where distributed input data is stored in several cloudlets, the computing method is the same, and the processing method corresponding to the lower response time can be determined when a request with simple operations in the second category happens. Next, we consider the case in which both cloudlets and the central cloud have a part of large volumes of input data. We assume that the complete input data is stored in two cloudlets and the central cloud, and the data size is m_1, m_2, and m_3, respectively. First, each cloudlet independently calculates its own T_i^c and $m_i a_i$ by Eq. 2 and determines the processing method according to the respective response time. Because the input data m_3 is processed by the central cloud, the corresponding processing time is $m_3 b$. Second, according to the distribution of the intermediate results, calculate the total response time T_i^b in each candidate aggregation node i that has partial intermediate results or is the central cloud. At last, the processing method and the aggregation node corresponding to the minimal response time are performed and selected.

Efficient Aggregation Schemes

The aggregation operation is a key step in performing complex mathematical operations because the required multiple inputs are geo-dispersed. Aggregating large amounts of multiple inputs may cause long delay for users. If the size of aggregated multiple inputs is big, the transmission time and cost can be reduced by an efficient aggregation scheme, which can improve the performance of AMF. We propose two types of aggregation scheme to support real-time and non-real-time geo-dispersed big data applications, respectively. Since real-time applications require smaller response time, the corresponding aggregation scheme is referred to as a dynamic aggregation scheme, which aims to reduce response time. On the other hand, for non-real-time applications, our proposed aggregation schemes aim to finish the requested tasks in a relatively longer time but with low cost, and the corresponding aggregation scheme is referred to as planned aggregation.

In the dynamic aggregation scheme, whenever a cloud node finishes extraction operation, it transfers the extracted multiple inputs and the corresponding size to other cloud nodes involved in the entire job. When all required multiple inputs are aggregated in a cloud node, the cloud node starts to perform analysis operation. After a cloud node finishes an analysis operation in the shortest time, it informs other cloud nodes with a state message including the original user request and final results. When a cloud node receives a state message, if it

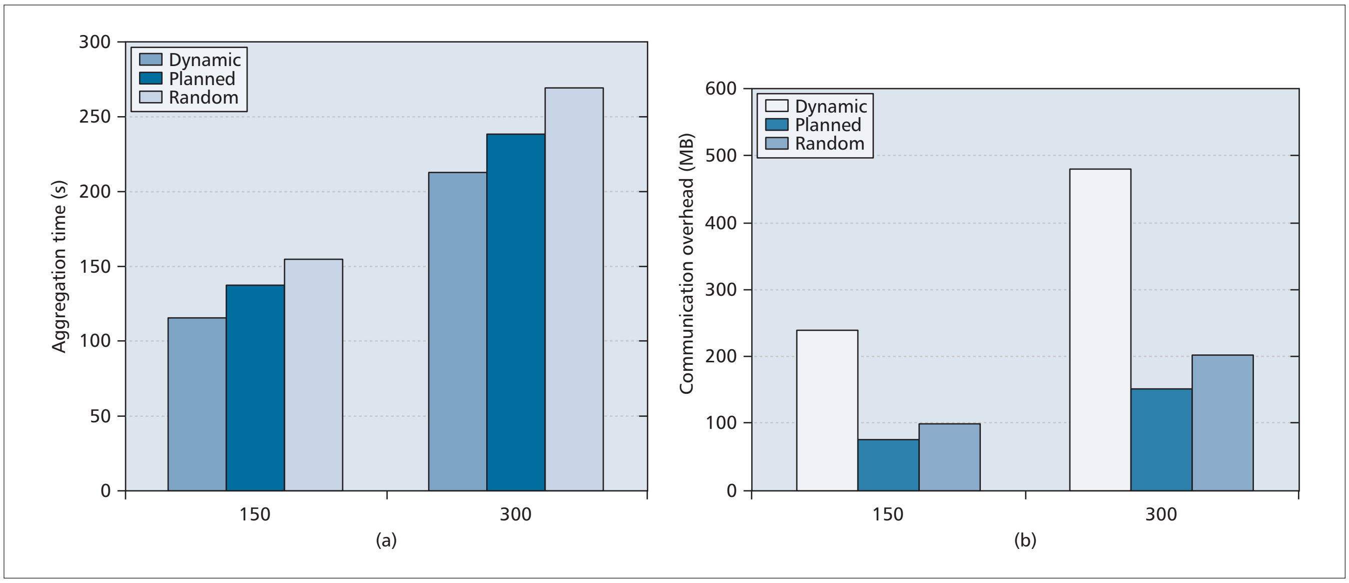

Figure 6. Simulation results of aggregation schemes: a) aggregation time; b) communication overhead.

is working on the same user request with the received state message, the cloud node ends all the operations caused by the same user request immediately. Due to the different computing capability of each cloud node, the respective time to extract required multiple inputs from a large amount of data is different. The dynamic aggregation scheme reactively carries the next task when the extraction of required multiple inputs in a cloud node is finished. Hence, the minimal aggregation time can be achieved by the dynamic aggregation scheme. The state message can save some communication and computation cost when the final results are obtained in the shortest time.

In the planned aggregation scheme, when all cloud nodes involved in a big job finish their own extraction operation, they start the planned aggregation scheme. In other words, after the slowest extraction operation is finished, the aggregation scheme starts. First, each cloud node transfers the data size of the extracted multiple inputs to other cloud nodes. Second, each cloud node finds the maximal size of extracted multiple inputs among all cloud nodes, and the cloud node that contains the maximal-size extracted multiple inputs is selected as the aggregation node. Third, each cloud node excluding the aggregation node transfers extracted inputs to the aggregation node. The planned aggregation scheme performs aggregation operation with minimal communication overhead. For non-real-time geo-dispersed big data applications, we assume that the analysis time and extraction time at each cloud node are acceptable to users.

We use NS-2 for simulation. Input data is distributed in two cloudlets and a central cloud. For evaluating performance of our aggregation schemes, we simulate two cases where the size of total extracted input data is 150 MB and 300 MB, respectively. In each case, we consider two types of distribution of extracted input data. For the first distribution of case 1, cloud1et 1 finishes its extract operations in 15 s, and the extracted input data is 25 MB. Cloud1et 2 finishes its extract operations in 25 s, and the extracted input data is 50 MB. The central cloud finishes its extract operations in 35 s, and the extracted input data is 75 MB. For the second distribution of case 1, the time to extract input data is the same with the first distribution. The extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 75 MB, 50 MB, and 25 MB, respectively. In case 2, the time to extract input data is the same as in case 1. For the first distribution of case 2, the extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 50 MB, 100 MB, and 150 MB, respectively. For the second distribution of case 2, the extracted input data in cloudlet 1, cloudlet 2, and the central cloud is 150 MB, 100 MB, and 50 MB, respectively. The link speed between cloudlets is 10 Mb/s corresponding to a LAN. The upstream link speed (i.e., the link from each cloudlet to the central cloud) is in the range of [1.3, 3.8] Mb/s, and the downstream link speed is in the range of [3.0, 4.1] Mb/s. We calculate aggregation time using the difference between the time when all extracted input data is aggregated in a node and the time when a request with complex operations happens. Figure 6 compares the aggregation time and communication overhead for three aggregation schemes (dynamic aggregation, planned aggregation, and random aggregation). The random aggregation scheme randomly selects a cloud node as the aggregation node when a request with complex operations arrives. In the random aggregation scheme, when a cloud node finishes its extract operations, it starts to transfer extracted input data to the selected aggregation node. The horizontal axis of Fig. 6 represents the size of the total extracted input data. The simulation results show that the dynamic aggregation scheme achieves the shortest aggregation time of the three schemes, and the planned aggregation scheme incurs less communication overhead than the others.

A Geo-Dispersed Big Data Application Based on the Proposed Framework

In this section, we consider a geo-dispersed big data application for analyzing vehicles to support an intelligent transportation system. The corresponding input data is a large amount of video surveillance data. The application is able to analyze traffic and identify a specific vehicle according to its color and license number. The video surveillance data is collected by high-definition cameras at roadsides and junctions. When a request happens, first AMF extracts the pictures from the original video surveillance data stored in several cloudlets according to the specified color of the request. Then AMF extracts the images from the extracted vehicle pictures according to the specified license number. At last, the extracted license images and the corresponding position information are aggregated and analyzed to identify the corresponding travel trajectory.

Conclusions

In this article, first we discuss the challenge in utilizing the mobile cloud to process geo-dispersed big data. Then we propose a novel and flexible framework based on MapReduce to support complex as well as simple operations on geo-dispersed big data. The proposed framework, AMF, employs the idea of parallel computing in MapReduce to extract multiple inputs for complex operations while being able to appropriately aggregate and analyze geo-dispersed big data. For simple operations, AMF adaptively utilizes collaboration among cloud nodes and MapReduce to efficiently process geo-dispersed big data. For complex operations on geo-dispersed big data, AMF uses different aggregation schemes to meet various application requirements. For real-time applications, minimizing response time to mobile users is achieved. For non-real-time applications, AMF makes a trade-off between response time and communication cost.

Acknowledgment

This research was partially supported by the National Basic Research Program of China (973 Program) under grant No. 2011CB302605 and the US National Science Foundation under grant CNS-1065444.

References

[1] J. Manyika *et al.*, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," McKinsey Global Inst., May 2011.
[2] S. Shekhar *et al.*, "Spatial Big-Data Challenges Intersecting Mobility and Cloud Computing," in *Proc.of 11th ACM Int'l. Wksp. Data Engineering for Wireless and Mobile Access*, Scottsdale, AZ, 2012, pp. 1–6.
[3] D. Huang *et al.*, "Secure Data Processing Framework for Mobile Cloud Computing," *Proc. IEEE INFOCOM Wksp. Cloud Computing*, Shanghai, China, 2011, pp. 614–18.
[4] B. Chun *et al.*, "Clonecloud: Elastic Execution between Mobile Device and Cloud," *Proc. 6th Conf. Comp. Sys.*, New York, NY, 2011, pp. 301–14.
[5] R. Yu *et al.*, "Toward Cloud-Based Vehicular Networks with Efficient Resource Management," *IEEE Network*, vol. 27, no. 5, Sept.–Oct. 2013, pp. 48–55.
[6] M. Felemban, S. Basalamah, and A. Ghafoor, "A Distributed Cloud Architecture for Mobile Multimedia Services," *IEEE Network*, vol. 27, no. 5, Sept.–Oct. 2013, pp. 20–27.
[7] D. Huang, T. Xing, and H. Wu, "Mobile Cloud Computing Service Models: A User-Centric Approach," *IEEE Network*, vol. 27, no. 5, Sept.–Oct. 2013, pp. 6–11.
[8] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, Jan. 2008, pp. 107–13.
[9] K. H. Lee *et al.*, "Parallel Data Processing with MapReduce: A Survey," *ACM SIGMOD Record*, vol. 40, no. 4, Dec. 2011, pp. 11–20.

Biographies

HONGLI ZHANG (zhanghongli@hit.edu.cn) received her B.S. degree in computer science from Sichuan University, China, in 1994, and her Ph.D. degree in computer science from Harbin Institute of Technology (HIT), China, in 1999. She is a professor in the Department of Computer Science and Technology at HIT and vice director of the National Computer Information Content Security Key Laboratory. Her research interests include network and information security, network measurement, and cloud computing.

QIANG ZHANG (zhangqiang@pact518.hit.edu.cn) received his B.E. degree in Information Security from HIT in 2009, and his M.S. degree from Harbin Engineering University, China, in 2012. He is now a Ph.D. student in the Department of Computer Science and Technology at HIT. His research interests include cloud computing and wireless networks.

ZHIGANG ZHOU (zhouzhigang@pact518.hit.edu.cn) received his B.S. and M.S. degrees in computer science from Dalian Jiaotong University, China, in 2004 and 2011. From 2011 to the present, he has been working as a Ph.D. candidate in the Department of Computer Science and Technology at HIT. His research interests include cloud computing and security.

XIAOJIANG DU (dxj@ieee.org) is currently an associate professor in the Department of Computer and Information Sciences at Temple University. He received his B.E. degree from Tsinghua University, China, in 1996, and his M.S. and Ph.D. degrees from the University of Maryland, College Park in 2002 and 2003, respectively, all in electrical engineering. His research interests are security, systems, wireless networks, and computer networks. He has published over 130 journal and conference papers in these areas, and has been awarded more than $3 million in research grants from the U.S. National Science Foundation and Army Research Office. He serves on the Editorial Boards of four international journals.

WEI YU (wyu@towson.edu) is an assistant professor in the Department of Computer and Information Sciences, Towson University, Maryland. Before that, he worked for Cisco Systems Inc. for nine years. He received his Ph.D. degree in computer engineering from Texas A&M University in 2008. His research interests include cyber security, computer networking, cyber-physical systems, and distributed systems.

MOHSEN GUIZANI [F] (mguizani@ieee.org) is currently a professor and vice president of graduate studies at Qatar University. Previously, he served as associate dean at Kuwait University; chair of the Computer Science Department at Western Michigan University; chair of the Computer Science Department at the University of West Florida; and director of graduate studies at the University of Missouri-Columbia. He received his B.S. (with distinction) and M.S. degrees in electrical engineering, and M.S. and Ph.D. degrees in computer engineering in 1984, 1986, 1987, and 1990, respectively, from Syracuse University, New York. His research interests include wireless communications and mobile computing, computer networks, smart grid, cloud computing, and security.